

**SMAI Project: Final Report**  
*on*  
**Named Entity Recognition**

***Submitted By:***

*Team Skylars*

***Team Members:***

*Nikita Gupta-201405527*

*Anamika Singh-201405534*

*Priyanka Bajaj-201405540*

*Ridhi Joshi-201405572*

***Guided By:***

*Nazrul Haque*

## **Table of Contents**

1. Introduction
2. Applications of NER
3. Data Set
4. Techniques Used
  - 4.1. Best Tag Approach
  - 4.2. HMM(Viterbi) Algorithm
  - 4.3. SVM Approach
5. Results
6. Conclusion
7. Code

## 1. Introduction

**Named-entity recognition (NER)** (also known as **entity identification**, **entity chunking** and **entity extraction**) is a subtask of information extraction that seeks to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

Full named-entity recognition is often broken down, conceptually and possibly also in implementations, as two distinct problems: detection of names, and classification of the names by the type of entity they refer to (e.g. person, organization, location and other).

NE recognises **entities** in text, and classifies them in some way, but it does not create templates, nor does it perform co-reference or entity linking, though these processes are often implemented alongside NE as part of a larger IE system.

NE is not just matching text strings with pre-defined lists of names. It only recognises entities which are being used as entities in a given context.

Now a day's machine learning approach is commonly used for NER because training is easy. There are various machine learning approaches for NER such as CRF (conditional Random Fields), MEMM (Maximum Entropy Markov Model), SVM (Support Vector Machine) and HMM (Hidden Markov Model) and dictionary based approach. Among all these HMM, being the most promising, has not been explored in its full potential for NER.

In this project we use the Hidden Markov Model (HMM) based approach of machine learning in detail to identify the named entities. The main idea behind the use of HMM model for building NER system is that it is language independent and we can apply this system for any language domain. In addition we also explore the SVM approach and Best Tag approach to perform comparative analysis over the results obtained.

Mostly the researchers use hybrid NER system which take advantages of both rule-based and statistical approaches so that the performance of NER system can be improved.

## 2. Applications of NER

NE involves identification of *proper names* in texts, and classification into a set of predefined categories of interest.

Three universally accepted categories: person, location and organisation.

Other common tasks: recognition of date/time expressions, measures (percent, money, weight etc), email addresses etc.

Other domain-specific entities: names of drugs, medical conditions, names of ships, bibliographic references etc.

- Named entities can be indexed, linked off, etc.
- Sentiment can be attributed to companies or products
- A lot of IE relations are associations between named entities
- A lot of IE relations are associations between named entities

- For question answering, answers are often named entities.

Other applications include, machine translation, text to speech synthesis, natural language understanding, Information Extraction, Information retrieval, question answering etc.

### 3. DataSet

We have used wikipedia data set, downloaded dataset from link :-  
<http://schwa.org/projects/resources/wiki/Wikiner>  
 Dataset is of the form :- word | Pos tag | NER tag.

Partition for training and testing is 70:30

***Procedure followed on the DataSet is as follows:***

#### Training

1. Collect a set of representative training documents
2. Label each token for its entity class or other (O)
3. Design feature extractors appropriate to the text and classes
3. Design feature extractors appropriate to the text and classes
4. Train a classifier to predict the labels from the data

#### Testing

1. Receive a set of testing documents
2. Run model inference to label each token
3. Appropriately output the recognized entities

### 4. Techniques Used

#### 4.1. Best Tag Approach

In this approach, tag is chosen on the basis of maximum emission probability.  
 For each word, probability of each tag is calculated. The tag for which probability is highest, that tag is assigned to the word.

#### *Greedy Inference*

- We just start at the left, and use our classifier at each position to assign a label
- The classifier can depend on previous labeling decisions as well as observed data

#### *Advantages:*

- Fast, no extra memory requirements
- Very easy to implement
- With rich features including observations to the right, it may perform quite well

#### *Disadvantage:*

Greedy. We may commit errors we cannot recover from.

## 4.2. HMM(Viterbi) Approach

HMM stands for Hidden Markov Model. HMM is a generative model. The model assigns the joint probability to paired observation and label sequence. Then the parameters are trained to maximize the joint likelihood of training sets.

Among all approaches, the evaluation performance of HMM is higher than those of others. The main reason may be due to its better ability of capturing the locality of phenomena, which indicates names in text.

We can define HMM in a formal way as follows:  $\lambda = (A, B, \pi)$ . Here, A represents the transition probability. B represents emission probability and  $\pi$  represents the start probability.  $A = a_{ij} = (\text{Number of transitions from state } s_i \text{ to } s_j) / (\text{Number of transitions from state } s_i)$ .  $B = b_j(k) = (\text{Number of times in state } j \text{ and observing symbol } k) / (\text{expected number of times in state } j)$ .

It means that the word occurs first in a sentence. Baum Welch Algorithm is used to find the maximum likelihood and posterior mode estimates for the HMM parameters. Forward-Backward Algorithm is used to find the posterior marginal's of all hidden state variables given a sequence of observations/emissions.

The **Viterbi algorithm** is a dynamic programming algorithm for finding the most likely sequence of hidden states – called the **Viterbi path** – that results in a sequence of observed events, especially in the context of Markov information sources and hidden Markov model. We are using HMM. A hidden Markov model (HMM) is a statistical Markov model in which the system being modeled is assumed to be a Markov process with unobserved (hidden) states.

Parameters of HMM Viterbi algorithm is following : Set of States, S where  $|S|=N$ . Here, N is the total number of states. Observations O where  $|O|=k$ . Here, k is the number of Output Alphabets.

Transition Probability, A

Emission Probability B

Initial State Probabilities  $\pi$

HMM may be represented as:  $\lambda = (A, B, \pi)$

Following formula for viterbi is used:-

$$P(X_1 = x_1 \dots X_n = x_n, Y_1 = y_1 \dots Y_{n+1} = y_{n+1}) \\ = \prod_{i=1}^{n+1} P(Y_i = y_i | Y_{i-2} = y_{i-2}, Y_{i-1} = y_{i-1}) \prod_{i=1}^n P(X_i = x_i | Y_i = y_i)$$

*Viterbi Inference*

- Dynamic programming or memoization.
- Requires small window of state influence (e.g., past two states are relevant).

*Advantage:*

- Exact: the global best sequence is returned.

*Disadvantage:*

- Harder to implement long-distance state-state interactions (but beam inference tends not to allow long-distance resurrection of sequences anyway).

### 4.3. SVM Approach

In machine learning, **support vector machines (SVMs, also support vector networks)** are supervised learning models with associated learning algorithm that analyze data used for classification and regression analysis. In addition to performing linear classification, SVMs can efficiently perform a non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces. When data are not labeled, supervised learning is not possible, and an unsupervised learning approach is required, which attempts to find natural clustering of the data to groups, and then map new data to these formed groups.

More formally, a support vector machine constructs a hyperplane or set of hyperplanes in a high- or infinite-dimensional space, which can be used for classification, regression, or other tasks. Intuitively, a good separation is achieved by the hyperplane that has the largest distance to the nearest training-data point of any class (so-called functional margin), since in general the larger the margin the lower the generalization error of the classifier.

*Implementation of SVM on our Data Set:*

For each word, we have found a vector of length 10 using word2vec, which will be our feature set.

## 5. Results

### *Best Tag Approach*

Accuracy: 86.45

Confusion Matrix:

| Tags   | I-LOC | B-ORG | I-PER | O      | I-MISC | B-MISC | I-ORG | B-LOC | B-PER |
|--------|-------|-------|-------|--------|--------|--------|-------|-------|-------|
| I-LOC  | 22570 | 562   | 1250  | 750    | 1378   | 368    | 2762  | 1761  | 4020  |
| B-ORG  | 4     | 1     | 2     | 4      | 4      | 0      | 17    | 2     | 6     |
| I-PER  | 1484  | 160   | 29494 | 431    | 1350   | 348    | 835   | 120   | 7945  |
| O      | 3457  | 97    | 541   | 817833 | 26592  | 1318   | 38055 | 818   | 13359 |
| I-MISC | 2228  | 361   | 2026  | 3471   | 19665  | 2181   | 4330  | 288   | 4704  |
| B-MISC | 3     | 0     | 0     | 1      | 36     | 17     | 3     | 0     | 28    |
| I-ORG  | 3030  | 1162  | 1053  | 1125   | 2338   | 407    | 15628 | 457   | 2738  |
| B-LOC  | 20    | 0     | 1     | 2      | 1      | 0      | 3     | 2     | 4     |
| B-PER  | 4     | 0     | 26    | 0      | 1      | 0      | 2     | 0     | 7     |

### *HMM Viterbi Approach*

Accuracy: 95.79

Confusion Matrix:

| Tags   | I-LOC | B-ORG | I-PER | O      | I-MISC | B-MISC | I-ORG | B-LOC | B-PER |
|--------|-------|-------|-------|--------|--------|--------|-------|-------|-------|
| I-LOC  | 27200 | 0     | 1245  | 4189   | 1247   | 0      | 1538  | 2     | 0     |
| B-ORG  | 4     | 0     | 1     | 6      | 4      | 0      | 25    | 0     | 0     |
| I-PER  | 1090  | 0     | 36411 | 2957   | 1154   | 0      | 552   | 0     | 3     |
| O      | 874   | 1     | 1107  | 895557 | 3280   | 2      | 1224  | 0     | 0     |
| I-MISC | 2025  | 1     | 2372  | 7124   | 24839  | 19     | 2873  | 1     | 0     |
| B-MISC | 1     | 0     | 1     | 4      | 72     | 3      | 7     | 0     | 0     |
| I-ORG  | 2179  | 7     | 1255  | 3314   | 2162   | 1      | 19012 | 0     | 0     |
| B-LOC  | 21    | 0     | 2     | 4      | 3      | 0      | 3     | 0     | 0     |
| B-PER  | 1     | 0     | 38    | 0      | 1      | 0      | 0     | 0     | 0     |

### *SVM Approach*

Accuracy: 92

Confusion Matrix:

| Tags   | I-LOC | B-ORG | I-PER | O     | I-MISC | B-MISC | I-ORG | B-LOC | B-PER |
|--------|-------|-------|-------|-------|--------|--------|-------|-------|-------|
| I-LOC  | 2022  | 0     | 597   | 408   | 150    | 0      | 108   | 0     | 0     |
| B-ORG  | 0     | 0     | 0     | 1     | 0      | 0      | 1     | 0     | 0     |
| I-PER  | 199   | 0     | 2483  | 118   | 74     | 0      | 39    | 0     | 0     |
| O      | 88    | 0     | 314   | 59360 | 52     | 0      | 38    | 0     | 0     |
| I-MISC | 342   | 0     | 522   | 814   | 255    | 0      | 61    | 0     | 0     |
| B-MISC | 2     | 0     | 0     | 1     | 0      | 0      | 3     | 0     | 0     |
| I-ORG  | 328   | 0     | 353   | 344   | 131    | 0      | 289   | 0     | 0     |
| B-LOC  | 20    | 0     | 1     | 2     | 0      | 0      | 3     | 0     | 0     |
| B-PER  | 0     | 0     | 3     | 0     | 0      | 0      | 1     | 0     | 0     |



## 6. Conclusion

| <i>Models (tested on smaller data)</i> |                   |                   |
|--|-------------------|-------------------|
| <b><i>Best Tag Approach</i></b>        | <b><i>HMM</i></b> | <b><i>SVM</i></b> |
| 90.59                                  | 98.89             | 92                |

We have most importantly compared two learning methods(HMM,SVM) and tested them on WikiNER data set.The comparison between these models is informative because of the different nature of the two leaning methods. In the case of the HMM the learning approach is generated, i.e. it makes use of positive examples to build a model of NE classes and then evaluates each unseen sentence to see how well each of the words fit the model. The SVM on the other hand, is a discriminative approach and makes use of both positive and negative examples to learn the distinction between the two classes. Another major difference is that the SVM outputs, a measure of distance from the classification function whereas the HMM uses the Viterbi algorithm to decode using maximum likelihood probabilities. Basically we expect the models to have quite different strengths and weaknesses and hopefully these can be complementary, allowing us eventually to combine the approaches to achieve a composite model.

## 7. Code

### *Best Tag Approach*

#### **best\_tag\_approach.py**

```
#!/usr/bin/python
from collections import defaultdict
import sys
import re

def convert(infilename, outfilename, showtag=True):
    fout = open(outfilename, 'wa+')
    fin=open(infilename,'r')
    lines=fin.readlines()
    for line in lines:
        #fout.write('\n\n\n')
        #print "Hello"
        #line = line.strip()
        pairs = line.strip().split(' ')
        for pair in pairs:
            # print pair
            temp = pair.split("|")
            if len(temp) == 3:
                # print "Word"
                word = temp[0]
                tag = temp[2]
                if showtag == True:
```

```

        fout.write(word+'\t'+tag+'\n')
    else:
        fout.write(word+'\n')
fout.close()

def evaluate(resultFile, keyFile):
    correct = 0
    n = 0
    fkey = open(keyFile, 'r')
    for l in open(resultFile, 'r'):
        n += 1
        if l.strip('\n') == fkey.readline().strip('\n'):
            correct += 1
    fkey.close()
    print (float(correct)/n) * 100

class HMM():

    def __init__(self, trainFileName, testFileName):
        self.ftrain = trainFileName
        self.ftest = testFileName

    def get_counts(self): # get counts for deriving parameters
        self.wordtag = defaultdict(int) # emission freqs
        self.unitag = defaultdict(int) # unigram freqs of tags
        self.bitag = defaultdict(int) # bigram freqs of tags
        self.tritag = defaultdict(int) # trigram freqs of tags

        tag_penult = ""
        tag_last = ""
        tag_current = ""

    for l in open(self.ftrain, 'r'):
        l = l.strip()
        if not l:
            tag_penult = tag_last
            tag_last = tag_current
            tag_current = ""
            # update sentence boundary case
            if tag_last != "" and tag_penult != "":
                # update bitag freqs
                self.bitag[(tag_last, tag_current)] += 1
                # update tritag freqs
                self.tritag[(tag_penult, tag_last, tag_current)] += 1
        else:
            word, tag = l.split('\t')
            tag_penult = tag_last
            tag_last = tag_current
            tag_current = tag
            # update emission freqs
            self.wordtag[(word, tag)] += 1
            # update unitag freqs
            self.unitag[tag] += 1
            # update bitag freqs
            self.bitag[(tag_last, tag_current)] += 1
            # update tritag freqs

```

```

        self.tritag[(tag_penult, tag_last, tag_current)] += 1
        # update starting bigrams
        if tag_last == " and tag_penult == "":
            self.bitag[(",")] += 1

def get_e(self, word, tag):
    return float(self.wordtag[(word, tag)] / self.unitag[tag])

def get_q(self, tag_penult, tag_last, tag_current):
    return float(self.tritag[(tag_penult, tag_last, tag_current)] / self.bitag[(tag_penult, tag_last)])

def get_parameters(self, method='UNK'): # derive parameters from counts
    self.words = set([key[0] for key in self.wordtag.keys()])
    if method == 'UNK':
        self.UNK()
    self.words = set([key[0] for key in self.wordtag.keys()])
    self.tags = set(self.unitag.keys())
    self.E = defaultdict(int)
    self.Q = defaultdict(int)
    for (word, tag) in self.wordtag:
        self.E[(word, tag)] = self.get_e(word, tag)
    for (tag_penult, tag_last, tag_current) in self.tritag:
        self.Q[(tag_penult, tag_last, tag_current)] = self.get_q(tag_penult, tag_last, tag_current)

def UNK(self):
    new = defaultdict(int)
    # change words with freq <5 into unknown words "<UNK>"
    for (word, tag) in self.wordtag:
        new[(word, tag)] = self.wordtag[(word, tag)]
        if self.wordtag[(word, tag)] < 5:
            new[('<UNK>', tag)] += self.wordtag[(word, tag)]
    self.wordtag = new

## baseline model, choosing the tag that maximizes emission probability
class HMM_Baseline(HMM):

    def run_UNK(self):
        self.get_counts()
        self.get_parameters()
        fout = open('test_out_baseline_UNK', 'w')
        best = {}
        # best tag for "<UNK>"
        pivot = 0
        besttag = ""
        for (word, tag) in self.E:
            if word == '<UNK>':
                if self.E[(word, tag)] > pivot:
                    pivot = self.E[(word, tag)]
                    besttag = tag
        best[('<UNK>')] = besttag
        #print '<UNK>', besttag
        i = 0 #counter, to visualize progress
        for l in open(self.ftest, 'r'):
            w = l.strip()
            if w:

```

```

        if w in best:
            fout.write(w+'\t'+best[w]+'\\n')
        else:
            pivot = 0
            besttag = ""
            if w not in self.words:
                fout.write(w+'\t'+best['<UNK>']+'\\n')
            else:
                for (word,tag) in self.E:
                    if word == w:
                        if self.E[(word,tag)] > pivot:
                            pivot = self.E[(word,tag)]
                            besttag = tag
                best[w] = besttag
                fout.write(w+'\t'+besttag+'\\n')
    else:
        fout.write("\\n")
    i += 1
    #if i%10000 == 0:
        #print i
fout.close()

```

```

def main():
    convert('./ENGLISH.test', 'test_key')
    convert('./ENGLISH.test', 'test', False)
    convert('./ENGLISH.train', 'train')
    BL_UNK = HMM_Baseline('train','test')
    BL_UNK.run_UNK()
    evaluate('test_out_baseline_UNK','test_key')

```

```

if __name__=="__main__":
    main()

```

### **matrix.py**

```

from sklearn.metrics import confusion_matrix

```

```

predicted_labels = []
actual_labels = []

```

```

#predicted_labels = tuple(open("test_out_baseline_UNK_app", 'r'))

```

```

with open("test_out_baseline_UNK_app", "r") as ins:
    for line in ins:
        predicted_labels.append(line.strip("\\n"))
ins.close()

```

```

with open("test_key_app", "r") as ins:
    for line in ins:
        actual_labels.append(line.strip("\\n"))
ins.close()

```

```

cm = confusion_matrix(actual_labels,predicted_labels,labels=["I-LOC", "B-ORG", "I-PER", "O", "I-MISC", "B-MISC","I-ORG", "B-LOC", "B-PER"])
print(cm)

```

## HMM(Viterbi) Approach

### hmm\_viterbi\_approach.py

```
#!/usr/bin/python
from collections import defaultdict
import sys
import re

def convert(infilename, outfilename, showtag=True):
    fout = open(outfilename, 'wa+')
    fin=open(infilename,'r')
    lines=fin.readlines()
    for line in lines:
        fout.write('\n\n\n')
        #print "Hello"
        #line = line.strip()
        pairs = line.strip().split(' ')
        for pair in pairs:
            # print pair
            temp = pair.split('|')
            if len(temp) == 3:
                # print "Word"
                word = temp[0]
                tag = temp[2]
                if showtag == True:
                    fout.write(word+'\t'+tag+'\n')
                else:
                    fout.write(word+'\n')
    fout.close()

def evaluate(resultFile, keyFile):
    correct = 0
    n = 0
    fkey = open(keyFile,'r')
    for l in open(resultFile,'r'):
        n += 1
        if l.strip('\n') == fkey.readline().strip('\n'):
            correct += 1
    fkey.close()
    print (float(correct)/n) * 100

class HMM():

    def __init__(self, trainFileName, testFileName):
        self.ftrain = trainFileName
        self.ftest = testFileName

    def get_counts(self): # get counts for deriving parameters
        self.wordtag = defaultdict(int) # emission freqs
        self.unitag = defaultdict(int) # unigram freqs of tags
        self.bitag = defaultdict(int) # bigram freqs of tags
        self.tritag = defaultdict(int) # trigram freqs of tags

        tag_penult = ""
        tag_last = ""
        tag_current = ""
```

```

for l in open(self.ftrain, 'r'):
    l = l.strip()
    if not l:
        tag_penult = tag_last
        tag_last = tag_current
        tag_current = ""
        # update sentence boundary case
        if tag_last != "" and tag_penult != "":
            # update bitag freqs
            self.bitag[(tag_last, tag_current)] += 1
            # update tritag freqs
            self.tritag[(tag_penult, tag_last, tag_current)] += 1
    else:
        word, tag = l.split('\t')
        tag_penult = tag_last
        tag_last = tag_current
        tag_current = tag
        # update emission freqs
        self.wordtag[(word,tag)] += 1
        # update unitag freqs
        self.unitag[tag] += 1
        # update bitag freqs
        self.bitag[(tag_last, tag_current)] += 1
        # update tritag freqs
        self.tritag[(tag_penult, tag_last, tag_current)] += 1
        # update starting bigrams
        if tag_last == "" and tag_penult == "":
            self.bitag[("","")] += 1

def get_e(self,word,tag):
    return float(self.wordtag[(word,tag)]/self.unitag[tag])

def get_q(self,tag_penult, tag_last, tag_current):
    return float(self.tritag[(tag_penult, tag_last, tag_current)]/self.bitag[(tag_penult, tag_last)])

def get_parameters(self, method='UNK'): # derive parameters from counts
    self.words = set([key[0] for key in self.wordtag.keys()])
    if method == 'UNK':
        self.UNK()
    self.words = set([key[0] for key in self.wordtag.keys()])
    self.tags = set(self.unitag.keys())
    self.E = defaultdict(int)
    self.Q = defaultdict(int)
    for (word,tag) in self.wordtag:
        self.E[(word,tag)] = self.get_e(word,tag)
    for (tag_penult, tag_last, tag_current) in self.tritag:
        self.Q[(tag_penult, tag_last, tag_current)] = self.get_q(tag_penult, tag_last, tag_current)

def tagger(self, outFileName, method='UNK'):
    # train
    self.get_counts()
    self.get_parameters(method)
    # begin tagging
    self.sent = []
    fout = open(outFileName, 'w')

```

```

count=0
counter=0
insidesent=0
insidenotl=0
elseinside=0
outl=0
for l in open(self.ftest, 'r'):
    l = l.strip()
    count=count+1
    if not l:
        #print l
        insidenotl=insidenotl+1
    if self.sent:
        # print "generate path for "+str(self.sent)+'\n'
        insidesent=insidesent+len(self.sent)
        path = self.viterbi(self.sent, method)
        # print path
        for i in range(len(self.sent)):
            counter=counter+1
            fout.write(self.sent[i]+'\\t'+path[i]+'\\n')
        self.sent = []
        else:
            elseinside=elseinside+1
            #fout.write('word+tag\\n')
        fout.write("\\n")
    else:
        outl=outl+1
        #fout.write('word+tag\\n')
        self.sent.append(l)
    print count,counter,insidenotl,insidesent,elseinside,outl
fout.close()

def viterbi(self,sent,method='UNK'):
    #print sent
    V = {}
    path = {}
    # init
    V[0,""] = 1
    path[""] = []
    # run
    #sys.stderr.write("entering k loop\\n")
    for k in range(1,len(sent)+1):
        temp_path = {}
        word = self.get_word(sent,k-1)
        ## handling unknown words in test set using low freq words in training set
        if word not in self.words:
            # print word
            if method=='UNK':
                word = '<UNK>'
            #sys.stderr.write("entering u loop "+str(k)+"\\n")
            for u in self.get_possible_tags(k-1):
                #sys.stderr.write("entering v loop "+str(u)+"\\n")
                for v in self.get_possible_tags(k):
                    V[k,u,v],prev_w = max([(V[k-1,w,u] * self.Q[w,u,v] * self.E[word,v],w) for w in
self.get_possible_tags(k-2)])
                    temp_path[u,v] = path[prev_w,u] + [v]

```

```

        path = temp_path
    # last step
    prob,umax,vmax = max([(V[len(sent),u,v] * self.Q[u,v,"],u,v) for u in self.tags for v in self.tags])
    return path[umax,vmax]

def get_possible_tags(self,k):
    if k == -1:
        return set([""])
    if k == 0:
        return set([""])
    else:
        return self.tags

def get_word(self,sent,k):
    if k < 0:
        return ""
    else:
        return sent[k]

def UNK(self):
    new = defaultdict(int)
    # change words with freq <5 into unknown words "<UNK>"
    for (word,tag) in self.wordtag:
        new[(word,tag)] = self.wordtag[(word,tag)]
        if self.wordtag[(word,tag)] < 5:
            new[('<UNK>',tag)] += self.wordtag[(word,tag)]
    self.wordtag = new

def main():
    convert('./ENGLISH.test', 'test_key')
    convert('./ENGLISH.test', 'test', False)
    convert('./ENGLISH.train', 'train')
    unk = HMM('train', 'test')
    unk.tagger('test_out_unk','UNK')
    evaluate('test_out_unk','test_key')

if __name__=="__main__":
    main()

```

### matrixhmm.py

```

from sklearn.metrics import confusion_matrix

predicted_labels = []
actual_labels = []

#predicted_labels = tuple(open("test_out_baseline_UNK_app", 'r'))

with open("test_out_unk_app", "r") as ins:
    for line in ins:
        predicted_labels.append(line.strip("\n"))
ins.close()

with open("test_key_app", "r") as ins:
    for line in ins:
        actual_labels.append(line.strip("\n"))

```



```
ins.close()
```

```
cm = confusion_matrix(actual_labels,predicted_labels,labels=["I-LOC", "B-ORG", "I-PER", "O", "I-MISC", "B-MISC", "I-ORG", "B-LOC", "B-PER"])
print(cm)
```

### *SVM Approach*

#### **svm.py**

```
import numpy as np
from sklearn import ensemble
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
import os, csv
import sys
import random

if __name__ == "__main__":

    traindata = np.genfromtxt("aij-wikiner-en-wp2_train.data", delimiter = ', ', skip_header=False)
    testdata = np.genfromtxt("aij-wikiner-en-wp2_test.data", delimiter = ', ', skip_header=False)
    print "Data Loading Completed :P"
    trainlabels = np.genfromtxt("aij-wikiner-en-wp2_train.labels", delimiter = ', ', skip_header=False)
    testlabels = np.genfromtxt("aij-wikiner-en-wp2_test.labels", delimiter = ', ', skip_header=False)
    print "Label Loading Completed"
    """print labels
    print traindata
    f = open('aij-wikiner-en-wp2.data', 'r')
    lenflines = f.readlines()
    lenf = len(lenflines)"""
    lentest = len(testlabels)

    train=[]
    for line in traindata:
        train.append(line)
    test=[]
    for line in testdata:
        test.append(line)

    trainLabels=[]
    for line in trainlabels:
        trainLabels.append(line)
    testLabels=[]
    for line in testlabels:
        testLabels.append(line)
    print "Loading Completed"

    clf = SVC(kernel='rbf', C = 1.0)
    clf.fit(train, trainLabels)
    print "Classification Completed :)"

    test_label=[]
    test_label=clf.predict(testdata)
    print "Classification Completed"
    labels=set(testLabels)
```

```

cm = confusion_matrix(testLabels, test_label, labels)
np.set_printoptions(precision=2)

print 'Confusion matrix'
print(cm)
"""for i in range(len(cm)):
    TP=0.0
    TN=0.0
    FP=0.0
    FN=0.0
    for j in range(len(cm)):
        for k in range(len(cm)):
            if(j == k and i == k):
                TP=TP+cm[j][k]
            elif(i == j):
                FP=FP+cm[j][k]
            elif(i == k):
                FN = FN+cm[j][k]
    acc= ((lentest-FP-FN)*100)/lentest;
    print "Accuracy of class",
    print labels[i],
    print acc
print "Done"
totCorrect=0
for i in range(len(cm)):
    for j in range(len(cm)):
        if(i == j):
            totCorrect=totCorrect+cm[j][k]
acc= (totCorrect*100)/lentest;
print "Accuracy of class",acc

```

### **dataSet.py**

```

#!/usr/bin/python
import sys
import re
from gensim import utils
from gensim.models.doc2vec import LabeledSentence
from gensim.models import word2vec

name_tag_list=[]
name_tag_set=[]
name_tag_dict={}
#fo = open("foo.txt", "rw+")
word_numeric_list=[]

model = word2vec.Word2Vec('~\GoogleNews-vectors-negative300.bin', size=300)
model = word2vec.Word2Vec.load_word2vec_format('~\GoogleNews-vectors-negative300.bin',
binary=True)

for line in open("aij-wikiner-en-wp2", "r").readlines():
    strs = re.sub(r'\s', ' ', line).split(' ')
    for values in strs:
        if(values.strip() == ""):
            continue
        word,pos_tag,name_entity_tag=values.split("|")
        #print word,

```

```

        temp =[]
        """print pos_tag,
        print name_entity_tag"""
        if word not in model:
            for i in range(300):
                temp.append(0)
        else:
            temp=model[word]
        word_numeric_list.append(temp)
        name_tag_list.append(name_entity_tag)
        #print len(temp)

name_tag_set=set(name_tag_list)
print name_tag_set
print len(word_numeric_list[0])
i=0
for tag in name_tag_set:
    name_tag_dict[tag]=i
    i=i+1

#-----Write Feature Matrix-----
fout=open('aij-wikiner-en-wp2.data','w')
for f in word_numeric_list:
    line=""
    for i in range(len(f)-1):
        line=line+str(f[i])+" "
    line=line+str(f[i])
    fout.write(line+"\n")
fout.close()

#-----Write Labels-----
fout=open('aij-wikiner-en-wp2.labels','w')
for f in name_tag_list:
    line=str(name_tag_dict[f])
    fout.write(line+"\n")
fout.close()

```